zfit

scalable pythonic fitting

Jonas Eschle on behalf of zfit
jonas.eschle@cern.ch

FNSNF
Swiss National Science Foundation

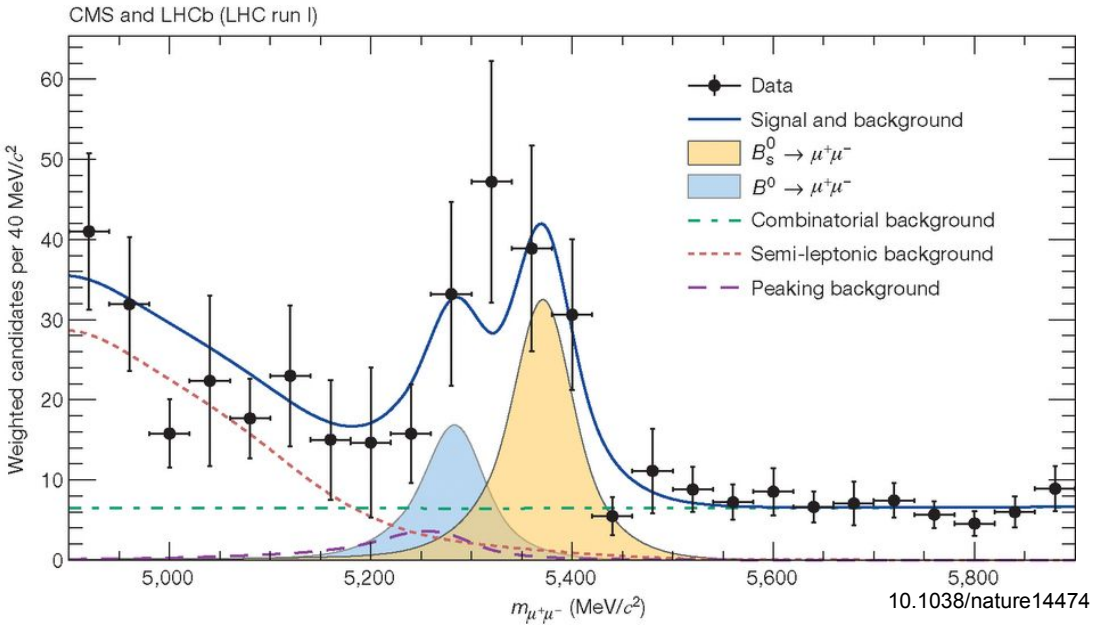
University of Zurich[UZH]

Model fitting in HEP

- Model from theory, depending on parameters
- Fit to data, extract parameter value
- Estimate uncertainty

Crucial step in most HEP analysis

Example: $B_s \rightarrow \mu\mu$



10.1038/nature14474

HEP fitting: RooFit

- C++ framework, well established
  - Used in vast majority of HEP analyses
- Why? Special needs in HEP:
  - Specific normalization (range)
  - Multiple dimensions
  - Custom models, compositions
  - Performance for large data, models
  - and more…

RooFit Shortcomings

- Limited in terms of customization and extendibility
- Sub-optimal scalability for ever larger datasets and modern computing infrastructure
- Isolated, aging ecosystem, no cutting-edge software
- HEP analysis moving towards Python
  - Offers huge scientific ecosystem
  - State-of-the-art packages shared with industry
  - RooFit Python bindings not (well) integrated

RooFit Shortcomings

- Limited in terms of customization and extensibility
- Sub-optimal scalability for ever larger datasets and modern computing infrastructure
- Isolated, aging ecosystem, no cutting-edge
- HEP analysis moving towards Python
- Offers huge scientific ecosystem

No feasible package in Python

- RooFit (Python bindings) not well integrated

zfit

What is `zfit`?

Model Fitting for HEP
(zfit core~RooFit core)

New library, codebase
(inspired by others)

Pure Python library

Community effort

What is zfit?

Model Fitting for HEP
(zfit core~RooFit core)

New library, codebase
(inspired by others)

TensorProb
mlfit
TensorFlow Probability
scipy
carl
probfit
RooFit
TensorFlow Analysis

Pythonic

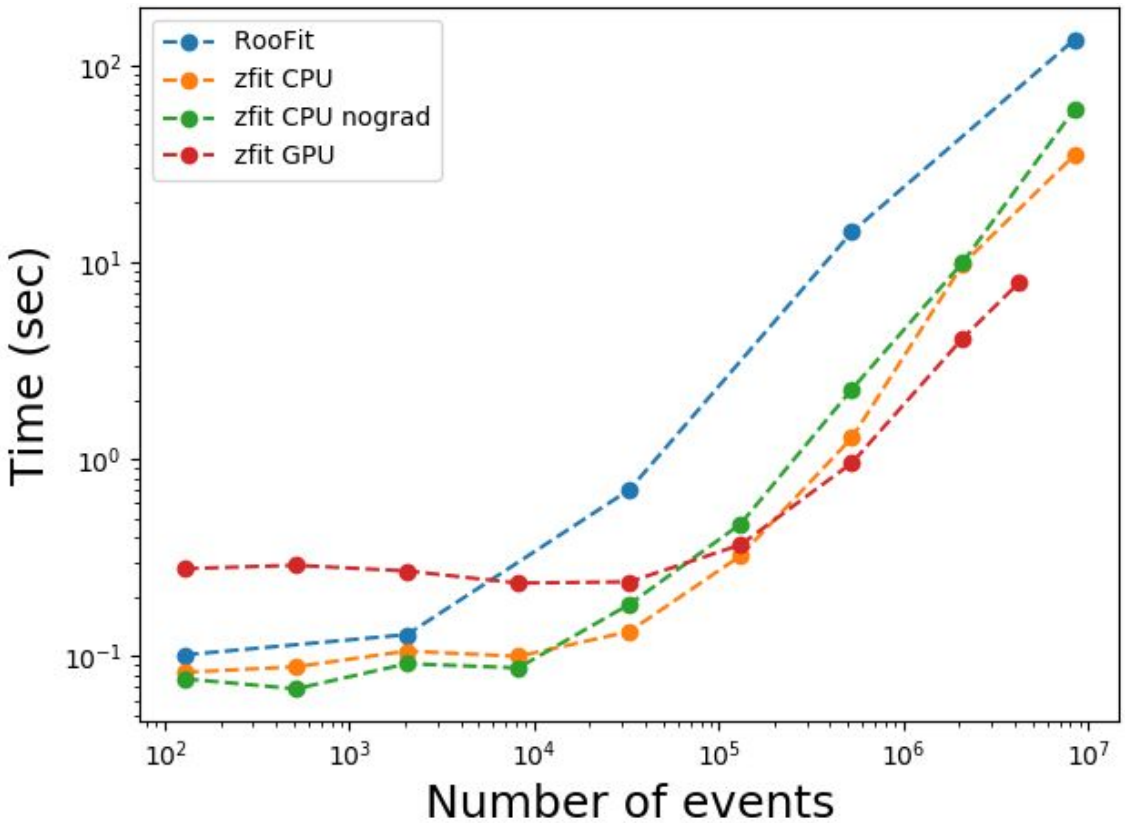
- Pure Python («pip install zfit»)
- Integrated into python ecosystem
    - Load ROOT files (HEP standard) with uproot (no ROOT dependence!)
    - Use Minuit (HEP standard) for minimization (iminuit)
    - Data preprocessing with Pandas DataFrame
    - Plotting with matplotlib
    - Higher level statistics tools
- Extendable classes, built for subclassing
  - e.g. custom models, loss, minimizers,...

Pure Python library...

Slow?!

Performance

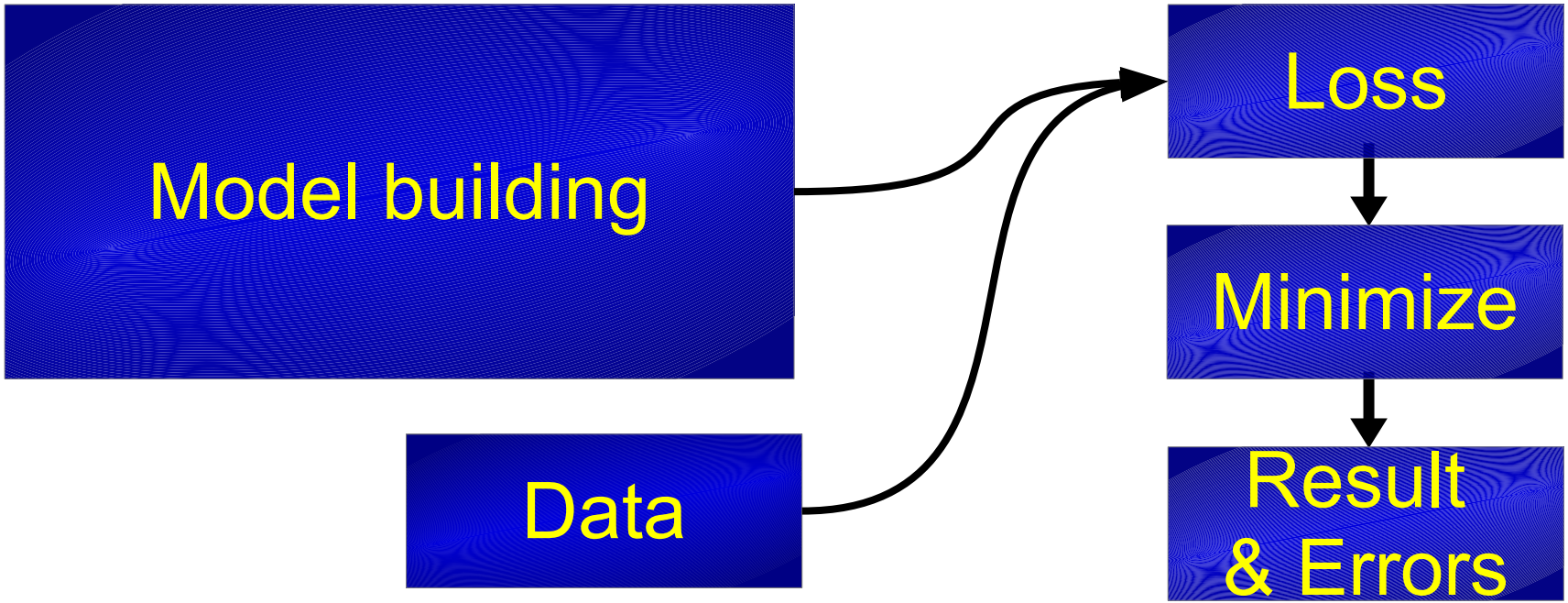
- Sum of 9 Gaussians
- Total 2 free param
- Toy study:
  measure fitting time



Scalable

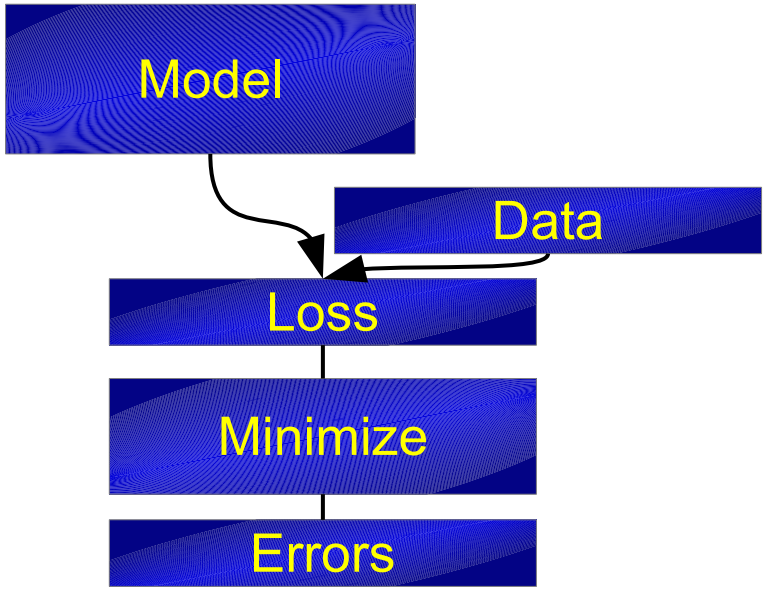
- TensorFlow **hidden** backend, uses graphs
  - numpy-like syntax
  - parallelization on CPU/GPU, analytic gradient,...
- Use with existing libraries
  - as used in multiple physics libraries and analyses: TensorFlow Analysis, …
  - Open source stack: TensorFlow Probability, Deep Learning,...

Workflow



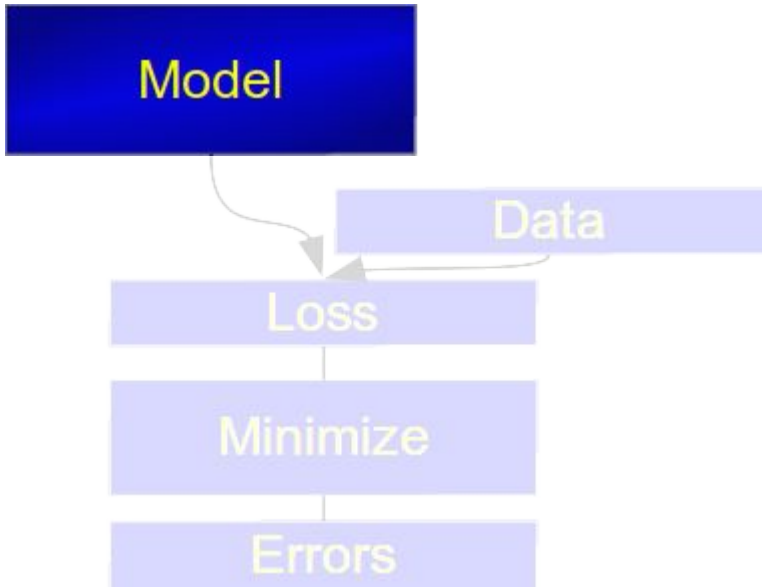
Complete fit

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```



Complete fit: Model

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```



Model Composition

Gaussian signal plus Exponential background shape

```
obs = zfit.Space("x", limits=(-10, 10))

mu =    zfit.Parameter("mu",          1,  -4,  6)
sigma = zfit.Parameter("sigma",       1, 0.1, 10)
lambd = zfit.Parameter("lambda",     -1,  -5,  0)
frac =  zfit.Parameter("fraction", 0.5,   0,  1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambd, obs=obs)

sum_pdf = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

parameters

models

Custom PDF

```python
from zfit import ztf


class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = x.unstack_x()
        alpha = self.params['alpha']

        return ztf.exp(alpha * data)
```

implement custom function

Custom PDF

```python
from zfit import ztf


class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = x.unstack_x()
        alpha = self.params['alpha']

        return ztf.exp(alpha * data)



custom_pdf = CustomPDF(obs=obs, alpha=0.2)

integral = custom_pdf.integrate(limits=(-1, 2))
sample   = custom_pdf.sample(n=1000)
prob     = custom_pdf.pdf(sample)
```

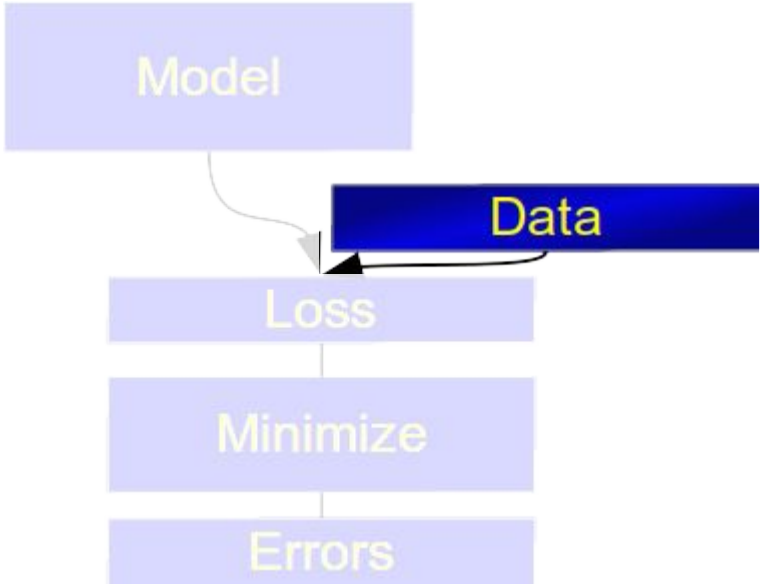
use functionality of model

LHCb Anomalies

```python
class P5pPDF(zfit.pdf.ZPDF):

    _PARAMS = ['FL', 'AT2', 'P5p']
    _N_OBS = 3

    def _unnormalized_pdf(self, x):
        FL = self.params['FL']
        AT2 = self.params['AT2']
        P5p = self.params['P5p']
        costheta_k, costheta_l, phi = ztf.unstack_x(x)

        sintheta_k = tf.sqrt(1.0 - costheta_k * costheta_k)
        sintheta_l = tf.sqrt(1.0 - costheta_l * costheta_l)

        sintheta_2k = (1.0 - costheta_k * costheta_k)
        sintheta_2l = (1.0 - costheta_l * costheta_l)

        sin2theta_k = (2.0 * sintheta_k * costheta_k)
        cos2theta_l = (2.0 * costheta_l * costheta_l - 1.0)

        pdf = (3.0 / 4.0) * (1.0 - FL) * sintheta_2k + \
              FL * costheta_k * costheta_k + \
              (1.0 / 4.0) * (1.0 - FL) * sintheta_2k * cos2theta_l + \
              -1.0 * FL * costheta_k * costheta_k * cos2theta_l + \
              (1.0 / 2.0) * (1.0 - FL) * AT2 * sintheta_2k * sintheta_2l * tf.cos(2.0 * phi) + \
              tf.sqrt(FL * (1 - FL)) * P5p * sin2theta_k * sintheta_l * tf.cos(phi)

        return pdf
```

Complete fit: Data

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```



Data

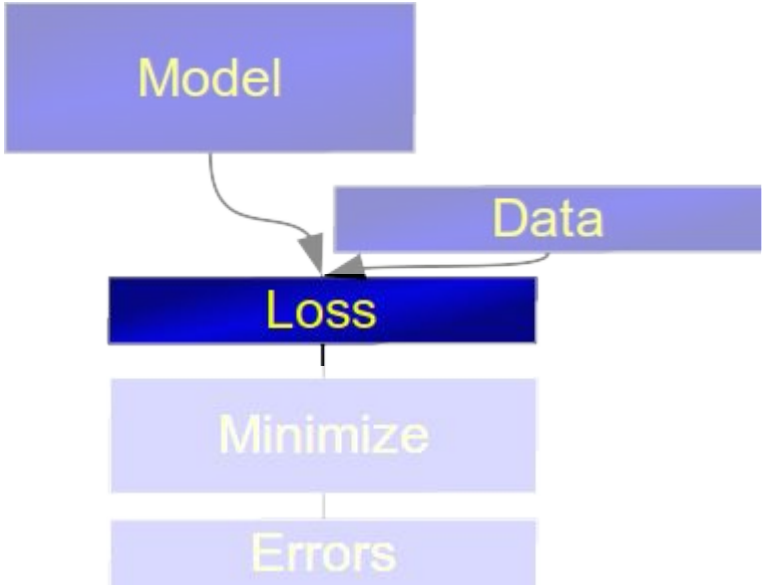
- From different sources
  - Numpy, Pandas, ROOT, …

- Sampled from a model (toy studies)

```
data = model.create_sampler(n_sample, limits=obs)
```

Complete fit: Loss

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```



Loss

- Builds loss from model and data
  - Likelihood etc
- Can be completely custom defined

  (see also talks tomorrow, e.g. [333] by Martina Ferrillo)
- Supports (arbitrary) constraints

```
param1_constr = zfit.constraint.GaussianConstraint(param1, mu=1.3, sigma=0.5)

param2_constr = zfit.constraint.SimpleConstraint(lambda: param2 ** 2, params=param2)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data,
                            constraints=[param1_constr, param2_constr])
```

Simultaneous fit

```
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
sigma1 = zfit.Parameter("sigma_one", 1., 0.1, 10)
sigma2 = zfit.Parameter("sigma_two", 1., 0.1, 10)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

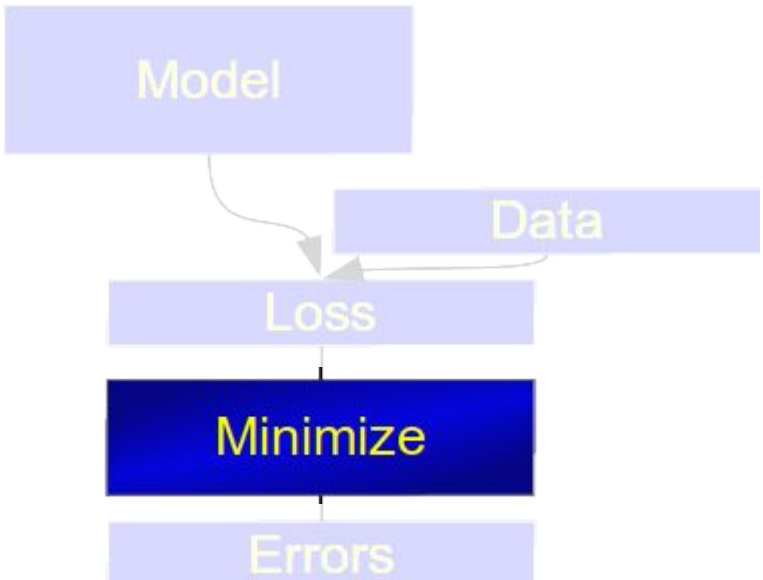
shared parameters

```
nll_simultaneous = zfit.loss.UnbinnedNLL(model=[gauss1, gauss2],
                                         data=[data1, data2])

nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

Completely equivalent

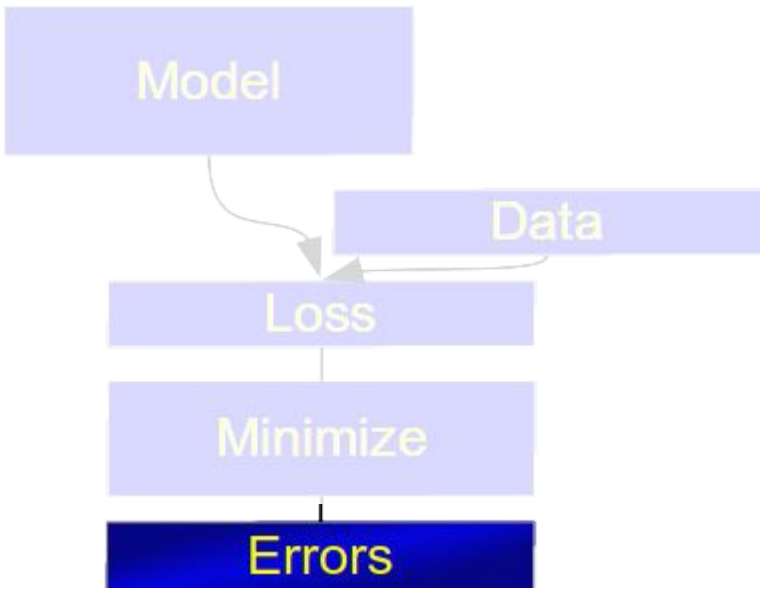
Complete fit: Minimization

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```



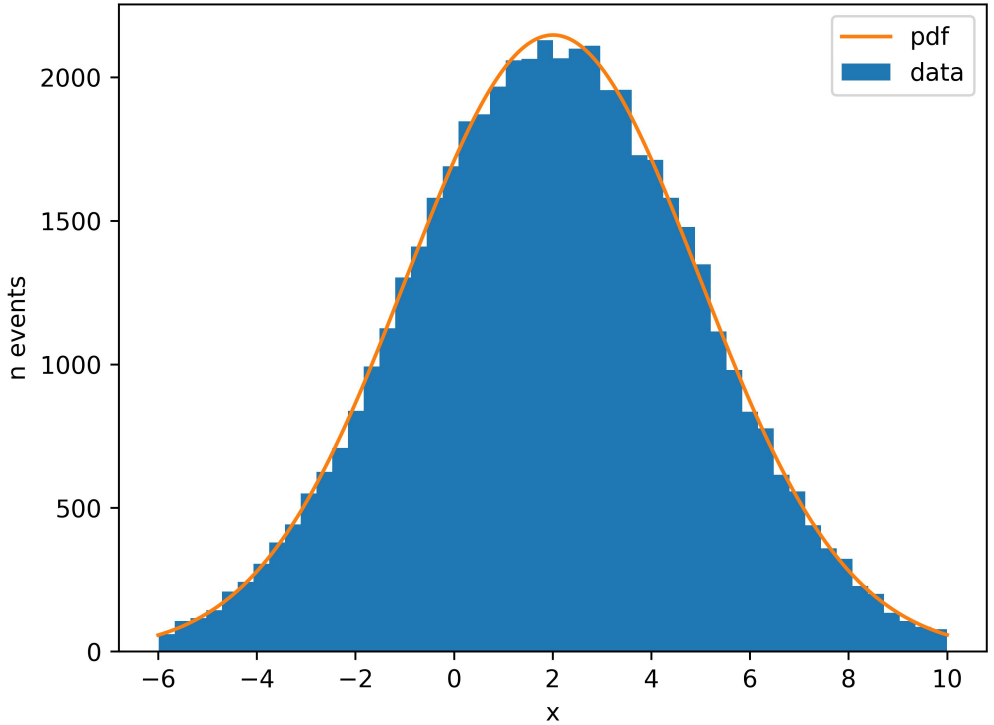
Complete fit: Result

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```

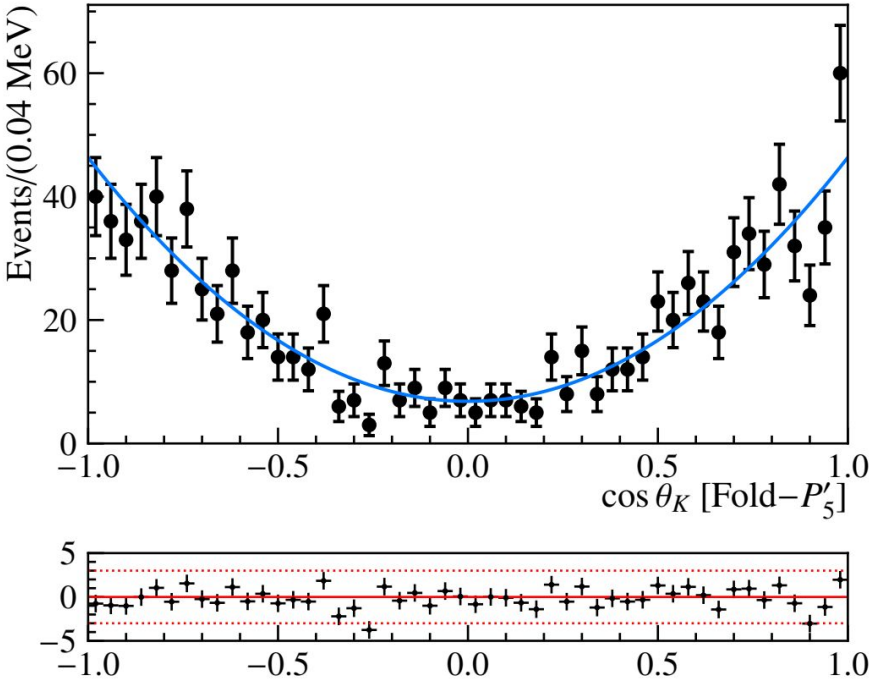


Complete fit: plots

Gaussian example



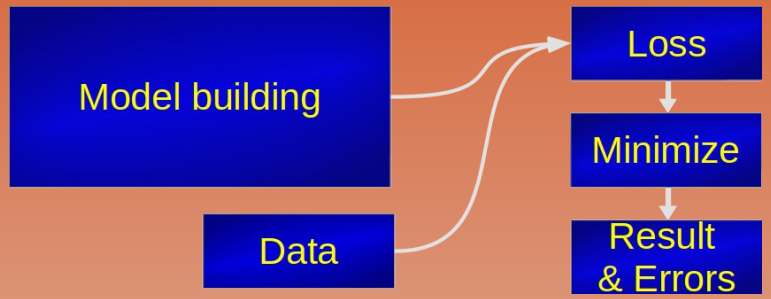
LHCb Anomalies



Summary

- Scalable Model fitting library for HEP
  - Well integrated into the Scientific Python Ecosystem
  - Customizable with well defined APIs
- Beta stage, usable! (already used in LHCb analyses)
  - *(Intentionally)* not feature complete, but API stabilizing
- Contributions in form of *feedback and criticism* very welcome
  - API, use-cases, bugs,…

scalable pythonic fitting



zfit

Backup Slides

https://zfit.github.io/zfit/

zfit@GitHub

Gitter channel

zfit@physik.uzh.ch

Join the discussion!

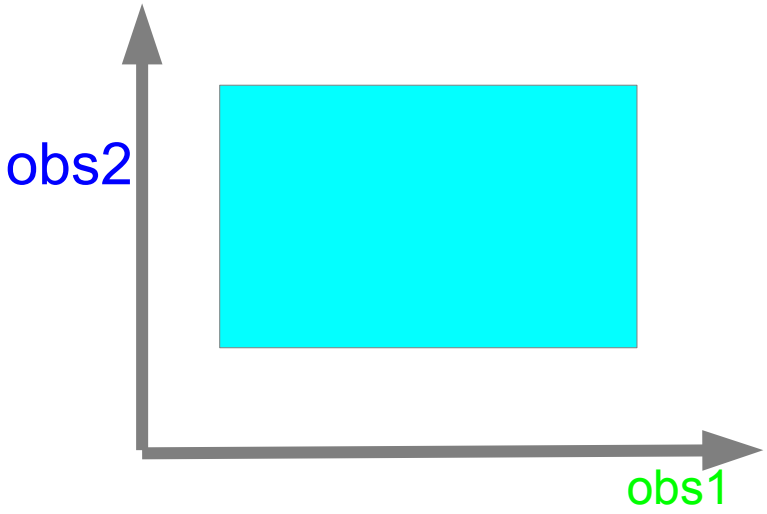
Scalable: TensorFlow

- Deep Learning framework by Google
- Modern, declarative graph approach
- Built for highly parallelized, fast communicating CPU, GPU, TPU,… clusters
- Built to use «Big Data»

Space

- A space is defined along one or several observables and usually has limits
  - «Named coordinate system»
- Each PDF, Data has a space
- Multiply two 2D PDFs with
  PDF1: [«obs1», «obs2»]
  PDF2: [«obs2», «obs3»]

  → 3D in [«obs1», «obs2», «obs3»]

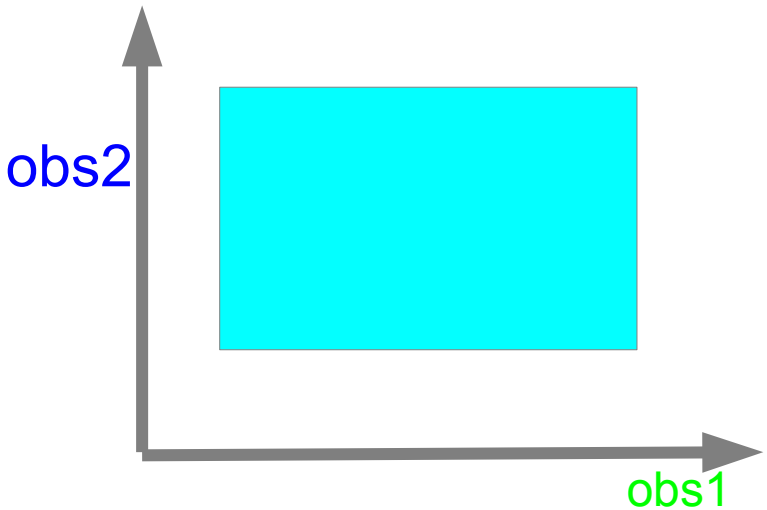


Complete fit: Space

```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```

Space

- A space is defined along one or several observables and usually has limits
  - «Named coordinate system»
- Each PDF, Data has a space
  - Normalization range, data cut,…
- Can be added, multiplied



Model building

```
obs = zfit.Space("x", limits=(-10, 10))

mu =    zfit.Parameter("mu",          1,  -4,  6)
sigma = zfit.Parameter("sigma",       1, 0.1, 10)
lambd = zfit.Parameter("lambda",     -1,  -5,  0)
frac =  zfit.Parameter("fraction", 0.5,   0,  1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambd, obs=obs)
```

parameters

models

Simultaneous fit

```
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
sigma1 = zfit.Parameter("sigma_one", 1., 0.1, 10)
sigma2 = zfit.Parameter("sigma_two", 1., 0.1, 10)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

shared parameters

```
nll_simultaneous = zfit.loss.UnbinnedNLL(model=[gauss1, gauss2],
                                         data=[data1, data2])

nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

Completely equivalent

Model, loss building

sum of two pdfs

```
sum_pdf = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

From classical

shared parameters

```
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

to more TensorFlow

simultaneous loss

```
nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

Model, loss building

Simple combinations

```
func_n = zfit.func.ZFunc(...)  # pseudo code
func = func_1 + func_2 * func_3
```

Composite Parameter

```
pdf = zfit.pdf.Gauss(mu=tensor1, sigma=4)
```

Custom Loss

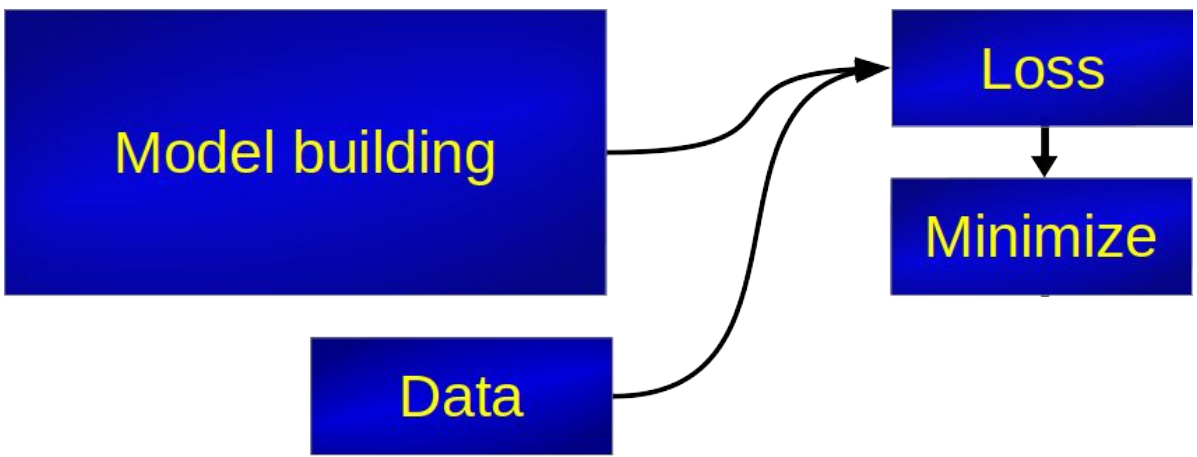
```
loss = zfit.loss.SimpleLoss(lambda: tensor_loss)
```

up to pure TensorFlow

=> use all of zfit functionality like minimizers

Scalable: TensorFlow

- Machine learning in a nutshell:
  - Build a model (a lot of matrix multiplications with simple non-linear functions in between) with 100k+ free parameters
  - Create a loss function (see how good/bad the predictions are)
  - Minimize it



Pythonic: statistics tool «lauztat»

- Author: Matthieu Marinangeli
- WIP, pre-beta
- Python statistics tool for limits, significance etc.
  (~ RooStats)
- lauztat on Github with example notebooks using `zfit`

Pythonic: «phasespace»

- Author: Albert Puig
- Python tool for n-body phasespace generation (~ TGenPhaseSpace)

Fitting: complete structure